SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

December 2021, Vol. 64 222103:1–222103:13
https://doi.org/10.1007/s11432-019-1519-6

# Text information aggregation with centrality attention

Jingjing GONG[1,2], Hang YAN[1,2], Yining ZHENG[1,2], Qipeng GUO[1,2], Xipeng QIU[1,2*] & Xuanjing HUANG[1,2]

[1]*School of Computer Science, Fudan University, Shanghai* 200433, *China;*
[2]*Shanghai Key Laboratory of Intelligent Information Processing, Fudan University, Shanghai* 200433, *China*

Received 29 April 2019/Accepted 29 July 2019/Published online 25 November 2021

**Abstract** A lot of natural language processing problems need to encode the text sequence as a fix-length vector, which usually involves an aggregation process of combining the representations of all the words, such as pooling or self-attention. However, these widely used aggregation approaches do not take higher-order relationships among the words into consideration. Hence we propose a new way of obtaining aggregation weights, called eigen-centrality self-attention. More specifically, we build a fully-connected graph for all the words in a sentence, then compute the eigen-centrality as the attention score of each word. The explicit modeling of relationships as a graph is able to capture some higher-order dependency among words, which helps us achieve better results in 5 text classification tasks and one SNLI task than baseline models such as pooling, self-attention, and dynamic routing. Besides, in order to compute the dominant eigenvector of the graph, we adopt a power method algorithm to get the eigen-centrality measure. Moreover, we also derive an iterative approach to get the gradient for the power method process to reduce both memory consumption and computation requirement.

**Keywords** information aggregation, eigen centrality, text classification, natural language processing, deep learning

**Citation** Gong J J, Yan H, Zheng Y N, et al. Text information aggregation with centrality attention. Sci China Inf Sci, 2021, 64(12): 222103, https://doi.org/10.1007/s11432-019-1519-6

## 1 Introduction

In the field of natural language processing (NLP), aggregating word representation vectors of variable lengths into one or several vectors is crucial to the success of various NLP tasks. This aggregation process prevails in text classification tasks [1, 2] and encode-based SNLI tasks [3].

There are a bunch of studies that try to represent text sequence as a fixed-size vector. Various methods differentiate from each other mostly in the way they encode the text sequence and aggregate word representations. In this paper, we mainly focus on aggregation operation, namely how to combine the sequence of vectors. Ref. [2] recursively encoded a sentence into a fixed size vector, and the process can be viewed as concurrent encoding and aggregating. Ref. [4] treated the hidden states of the last time step in a variant of LSTM as the encoded vector of a sentence. Ref. [5] combined convolutional neural networks and max-pooling to solve text classification tasks. For the purpose of alleviating the remembering burden on encoding layers, Ref. [6] designed an attention mechanism to calculate a weight for each encoded state. The aforementioned aggregation methods lack integrating global information into the decision of the weights, and then Ref. [7] enhanced the representational ability of aggregated vector by dynamic routing, which is iteratively refining the routing weights taking more context information into consideration after each iteration.

Among several aggregation methods, the self-attention mechanism [6, 8] shows its advantage of flexibility and effectiveness. However, the self-attention assigns the weight to each word according to its

* Corresponding author (email: xpqiu@fudan.edu.cn)

 

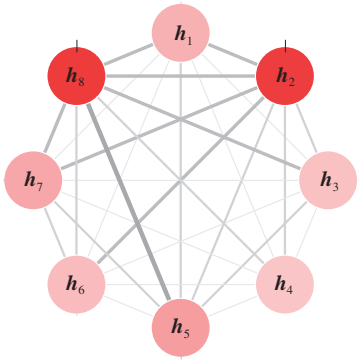


**Figure 1** (Color online) Eigenvector centrality of a fully-connected word graph. Darker color of a node indicates it is more important, the thickness of the edge represents the connected strength. The centrality is computed based on the connected relationship among all the words.

own representation and a query vector, and ignores the impact of the contextual words. Intuitively, the importance of a word in a sentence should depend on the relationships among all the words.

In this paper, we propose a new method to calculate the attention weight by explicitly modeling an intensive relationship between words in the sequence. We first construct a fully-connected word graph in which the edges are the relationships between words, and vertices are words in the sequence. We then apply the centrality algorithm on this graph and then the importance score of each vector will emerge. The "centrality" is the importance of each node in a graph. The assumption is that each node's centrality is the sum of the centrality values of the nodes that it is connected to. Figure 1 gives an intuitive illustration of the centrality of a fully-connected graph.

The contribution of this paper can be summarized as follows:

(1) We have identified that most aggregation methods are independent of contextual words, and thus propose a more context-aware approach to obtain attention weights, which considers an intense interaction between words.

(2) We utilize the power method to get the dominant eigenvector of the graph, but constructing a computation graph during the iterative process of the power method makes the back propagation very low-efficient. Therefore, we derive an approach to approximate the gradient of the power method which will significantly reduce memory consumption and the back-propagation computation complexity.

(3) We justify the use of "stop gradient before convergence" trick to reduce the memory utility while still utilizing auto-differentiation frameworks.

## 2 Background

### 2.1 Problem definition

Commonly, the sentence encoding process aims to find a fix-length vector to represent the meaning of a variable-length sentence, which consists of the following three layers.

**Embedding layer.** Given a sentence $S = x_1, x_2, \ldots, x_n$, most sentence encoding models first transform each word in $S$ into $d$ dimensional dense vectors. Thus, the sentence is represented by a matrix into $X = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n] \in \mathbb{R}^{d \times n}$. Some pre-trained embedding, such as word2vec [9] or GloVe [10], can be used for a good initialization. However, these pre-trained embeddings lack the awareness of words' context, ELMO [11] solves this problem by generating word embeddings by the language model. To have fair comparison with previous study, we use static embedding Glove in this paper.

**Fusion layer.** Fusion layer is responsible for modeling the semantic compositions between words, RNN, CNN and Transformer are universally adopted. No matter what kind of structures are applied, the basic form of fusion layer can be formalized as

$$\boldsymbol{h}_1, \boldsymbol{h}_2, \ldots, \boldsymbol{h}_n = \text{fuse}(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n), \tag{1}$$

$$\boldsymbol{H} = [\boldsymbol{h}_1, \boldsymbol{h}_2, \ldots, \boldsymbol{h}_n]. \tag{2}$$

During the fusion phase, $x_t$, as well as all other word representations should have an impact on the value of $h_t$. In this paper, BiLSTM, which not only considers the forward contextual correlation but also accounts for information flow from the reverse direction, serves as the encoder layer. We concatenate the forward hidden state and the backward hidden state in the last layer of the BiLSTM.

**Aggregation layer.** Different sentences vary in length, the aggregation of variable-length representations $H$ into a fixed-length vector is necessary in most NLP tasks. A general aggregation method is to assign a weight $\alpha_i$ to each word, then the summarized vector $\bar{\boldsymbol{h}} \in \mathbb{R}^d$ is

$$\bar{\boldsymbol{h}} = \sum_{i=1}^{n} \alpha_i \boldsymbol{h}_i, \tag{3}$$

where the weight $\alpha_i$ can be determined statically or dynamically according to different strategies.

Following are four commonly used aggregation layers.

(1) Max-pooling takes the maximum value of encoded representations in each dimension, it can be expressed as

$$\bar{\boldsymbol{h}} = \max([\boldsymbol{h}_1, \boldsymbol{h}_2, \ldots, \boldsymbol{h}_n]). \tag{4}$$

(2) Average pooling sets the weight of each word to $\alpha_i = \frac{1}{n}$. The average pooling is an aggregation method that feature vectors are weakly involved in determining the process of aggregation. Each word can only influence how it is aggregated by its quantity, because the weight of each word is inverse proportional to its length.

(3) Self-attention dynamically assigns a weight to each word according to its representation [6,8], then sums up all the words according to their weights. A trainable query $\boldsymbol{q}$ of the same dimensions as the encoded representation is used to calculate this weight. Followings are self-attention's formula:

$$\alpha_i = \frac{\exp(\boldsymbol{q}^{\mathrm{T}} \boldsymbol{h}_i)}{\sum_{j=1}^{n} \exp(\boldsymbol{q}^{\mathrm{T}} \boldsymbol{h}_j)}. \tag{5}$$

Though being more complex than average pooling, the score of self-attention also neglects other words in the sentence, and the only interactions among the words are the softmax normalization process.

(4) Dynamic routing is a mechanism that transfers the information of $H$ into a certain number of vectors $V$. Normally there are two manners to further exploit vectors $V$, one is to treat each vectors as the representation for each classes [12], the other is to concatenate vectors in $V$ into a fixed-length vector [13]. The detail dynamic routing process can be found in [12].

### 2.2 Eigenvector centrality

The centrality measure is a kind of quantitative symbol of influence (importance) value of a node in a network. There exist several popular centralities: (1) In-degree centrality, a node's in-degree centrality is decided by the number of nodes pointing to it. (2) Closeness centrality, to get a node's closeness centrality, first sums up distances between the node and its connected nodes, then computes the sum's reciprocal. (3) Betweenness centrality is the number of times a node appears in the shortest path between other two nodes. (4) Eigenvector centrality, which we will describe in Section 3. Refs. [14–17] utilize eigenvector centrality to capture intrinsic neural architectures through fMRI data. Eigenvector centrality is regarded as an indicator of global prominence of a feature in [18].

In this study, we believe the eigen-centrality of a word in a sentence can indicate the importance of this word, and we take its eigen-centrality value as its weight to aggregate the sentence.

## 3 Eigen-centrality self-attention

Given a sentence with its representation $H = [\boldsymbol{h}_1, \boldsymbol{h}_2, \ldots, \boldsymbol{h}_n] \in \mathbb{R}^{d \times n}$, the aggregation is to assign a weight $\alpha_i$ to each word. Intuitively, the weight of each word should be determined by the relations between all the words in the sentence.

Therefore, we construct an adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ to describe the connectivity between words in the vector list. Each element in $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is formulated as follows:

$$A_{ij} = f(\boldsymbol{h}_i, \boldsymbol{h}_j), \tag{6}$$

where $A_{ij}$ is the $i$th row $j$th column of $\boldsymbol{A}$, function $f$ is a scalar connectivity function that its result is strictly positive. Connectivity function $f$ should be designed to be trainable functions that are able to capture the relationship between a pair of components and describe it as connectivity intensity. In this paper, the $f$ is chosen to be a two layer fully connected neural network followed with a column-wise softmax normalization to make the adjacency matrix $\boldsymbol{A}$ be a left stochastic matrix.

### 3.1 Calculating attention score via eigen-centrality

Given the adjacency matrix $\boldsymbol{A}$ of a sequence of words, where $\boldsymbol{A}_{ij}$ denotes a connection strength from words $i$ and $j$. A larger value of $\boldsymbol{A}_{ij}$ indicates a stronger relation between words $i$ and $j$.

The importance of a word can be measured by centrality. Here, we use the eigenvector centrality to calculate the importance of a word with respect to other words. A high score of a word suggests that there are large connected weights between it and the other important nodes. A well-known application of eigenvector centrality is Google's PageRank algorithm.

The relative centrality score of $i$th word can be defined as

$$\alpha_i = \frac{1}{\lambda} \sum_j A_{i,j} \alpha_j, \tag{7}$$

where $\alpha_i$ denotes the relative importance measure of $i$th vertex, $\lambda$ is a constant, note that both $\alpha_i$ and $\lambda$ are unknowns that need to be solved. Let $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_n)^{\mathrm{T}}$, Eq. (7) can be formulated as vector notation:

$$\boldsymbol{A}\boldsymbol{\alpha} = \lambda \boldsymbol{\alpha}, \tag{8}$$

where $\boldsymbol{\alpha}$ happens to be an eigenvector of adjacency matrix $\boldsymbol{A}$, note that only the eigenvector with all its values positive will satisfy the requirement. And according to Perron-Frobenius theorem [19, 20], when all entries $\boldsymbol{A}_{ij}$ in the adjacency matrix $\boldsymbol{A}$ are strictly positive, then there exists and only exists one eigenvector $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)^{\mathrm{T}}$ of $\boldsymbol{A}$ with a dominant eigenvalue $\lambda$ such that all components of $\boldsymbol{\alpha}$ are positive: $\boldsymbol{A}\boldsymbol{\alpha} = \lambda\boldsymbol{\alpha}, \forall i, \alpha_i > 0$. It is known in the literature under many variations, such as the Perron vector, Perron eigenvector, Perron-Frobenius eigenvector, leading eigenvector, or dominant eigenvector.

### 3.2 Obtaining dominant eigenvector via power method

Now we need to solve (8) to get the dominant vector $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)^{\mathrm{T}}$. Because we only need the dominant eigenvector, we apply the power method algorithm to obtain the dominant eigenvector of the adjacency matrix with moderate computational cost.

**Algorithm 1** Power method

Input: connectivity matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$, initialize random vector $\boldsymbol{y} = \boldsymbol{z} \in \mathbb{R}^n$;
Repeat $\boldsymbol{\alpha} = \frac{\boldsymbol{y}}{\|\boldsymbol{y}\|_2}$, $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{\alpha}$, $\theta = \boldsymbol{\alpha}^{\mathrm{T}}\boldsymbol{y}$;
Until $\|\boldsymbol{y} - \theta\boldsymbol{\alpha}\|_2 \leqslant \epsilon\,|\theta|$;
//Stop when converge.
Output: dominant eigenvalue $\lambda = \theta$ and corresponding eigenvector $\boldsymbol{\alpha}$.

Ref. [21] made convergence analysis on power method convergence and showed that the power method good at approximating eigenvalue and eigenvectors. In the scope of this paper, the adjacency matrix is strictly positive. With the algebrac and geometric multiplicities of Perron-root both being 1, unless the initial vector $\boldsymbol{z}$ is strictly perpendicular with respect to the dominant eigenvector, a randomly initialized start vector could converge to the dominant eigenvector. Thus we initialize the random $\boldsymbol{z}$ with all positive components to prevent a perpendicular initialization. We will discuss the initialization of this initial vector in detail in Appendixes A.2 and A.3.

### 3.3 Gradient computation for power method

With auto-differentiation tools, we can easily compute the gradient for a normal operation. But the power method needs to run for ideally infinite iterations to get an accurate approximation of gradient, to use auto-differentiation we need to store intermediate states for every iteration step, and the backward is also to be computed as many steps as forward iteration. In an ideal case where we run the power method

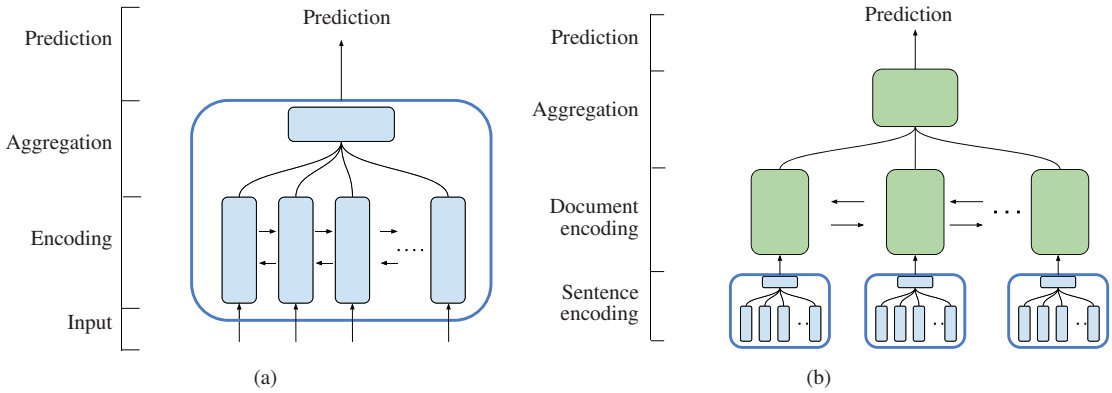


**Figure 2** (Color online) Text classification architectures. (a) is the architecture for SST-1 and SST-2 datasets; (b) is the architecture for IMDB, Yelp datasets. They share the same sequence encoding model which is BiLSTMs and the aggregation method — eigen-centrality attention.

for a large number of steps, the memory consumption would be unbearable, and the backward process would also double the computation as needed in forward iteration.

In this paper, we prove Theorem 1 that when the power method runs for ideally infinite steps, initial vector $\boldsymbol{z}$ receives neglectable amount of gradient which means the initial $\boldsymbol{z}$ will not affect the result of gradient, this also indicates that gradient received in early steps is neglectable, which further indicates that only last few steps of iteration matters. This suggests that when convergence criteria have reached (we call it convergence phase), we can run extra steps to obtain a good approximation of the gradient (we call it gradient iteration phase), also because states in the early step have little influence on the final gradient, in convergence iteration phase we can iterate without storing any intermediate steps.

**Theorem 1.** Given an adjacency matrix $\boldsymbol{A}$ with its component $A_{ij} > 0, \forall i, j$, randomly initialize $\boldsymbol{z}$ with positive entries as the initial vector of power method for infinite steps. The partial derivative of the output eigenvector $\boldsymbol{\alpha}$ with respect to $\boldsymbol{z}$ (or early steps of $\boldsymbol{\alpha}$) is $\mathbf{0}$.

Because we can approximate gradient with last few steps of iteration, we know that the eigenvector would have been long converged, which means we even do not need to store the intermediate state in gradient phrase iterations, because every intermediate eigenvector would be approximately the same, and the adjacency matrix $\boldsymbol{A}$ is also constant.

Thus, we are also introducing a reverse iteration approach (Eq. (9)) to calculate the gradient of power method, in which if the approximation of the dominant eigenvector is good enough, we can bound the error of the approximated gradient to a small margin. Moreover, no intermediate states will be kept, because after infinite steps the intermediate state would be the same since it would have long been converged to the dominant eigenvector. We will prove Theorem 2 in Appendix A.1.

**Theorem 2.** Given an adjacency matrix $\boldsymbol{A}$ with all its components $A_{ij} > 0, \forall i, j$, it is corresponding dominant eigenvalue $\lambda$ and corresponding eigenvector $\boldsymbol{\alpha}$ computed with the power method algorithm for infinite steps. The derivative of loss $\mathcal{L}$ with respect to $\boldsymbol{A}$ can be computed as

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{A}} = \lim_{N \to \infty} \sum_{k=0}^{N} \boldsymbol{\gamma}^{\mathrm{T}} (\boldsymbol{J}_{\boldsymbol{\alpha}})^k \boldsymbol{J}_{\boldsymbol{A}}, \tag{9}$$

where $\boldsymbol{\gamma}$ is the partial derivative of $\mathcal{L}$ with respect to last step of $\boldsymbol{\alpha}$, $\boldsymbol{J}_{\boldsymbol{\alpha}}$ is the jacobian matrix of $\boldsymbol{\alpha}$ with respect to previous step of $\boldsymbol{\alpha}$, $\boldsymbol{J}_{\boldsymbol{A}}$ is the jacobian matrix of $\boldsymbol{\alpha}$ with respect to previous step of $\boldsymbol{A}$ ($\boldsymbol{J}_{\boldsymbol{\alpha}}$ and $\boldsymbol{J}_{\boldsymbol{A}}$ depend only on $\boldsymbol{A}$, $\boldsymbol{\alpha}$ and $\lambda$).

## 4 Application to NLP tasks

There are three kinds of tasks in this paper, sentence-level classification, document-level classification, and encoder-based texture entailment task. The sequence encoding is exactly the same but with a different network structure.

**Flatten sentence classification architecture.** In sentence-level classification, we adopt a flatten structure which encodes the word embedding sequence with a BiLSTMs followed with an aggregation

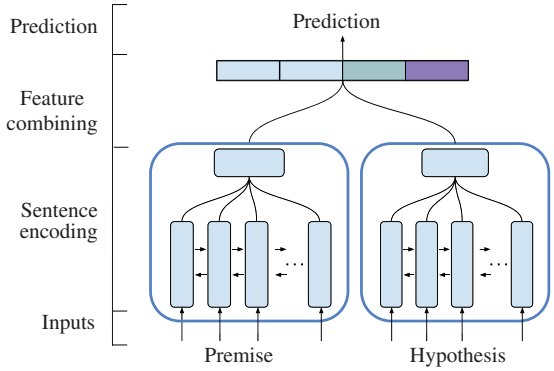


**Figure 3** (Color online) NLI architecture. The encoder is the same as in Figure 2(a).

**Table 1** Statistics of the five classification datasets used in this paper

| Dataset | Type | Train size | Development size | Test size | Classes | Averaged length | Vocabulary size |
|---|---|---|---|---|---|---|---|
| Yelp 2013 | Document | 62522 | 7773 | 8671 | 5 | 189 | 29.3k |
| Yelp 2014 | Document | 183019 | 22745 | 25399 | 5 | 197 | 49.6k |
| IMDB | Document | 67426 | 8381 | 9112 | 10 | 395 | 61.1k |
| SST-1 | Sentence | 8544 | 1101 | 2201 | 5 | 18 | 16.3k |
| SST-2 | Sentence | 6920 | 872 | 1821 | 2 | 19 | 14.8k |
| SNLI | NLI-task | 549k | 9.8k | 9.8k | 3 | 11 | 36k |

operation. The aggregation operation in this paper is the proposed eigen-centrality attention. The architecture schematic is shown in Figure 2(a).

**Hierarchical document classification architecture.** In document-level classification, we take a hierarchical approach [8]. We first encode sentences and aggregate them as sentence embeddings with BiLSTMs followed with the proposed eigen-centrality attention, and then encode the sentence representations as document representations with BiLSTMs followed with an eigen-centrality attention. The hierarchical classification architecture is shown in Figure 2(b).

**NLI architecture.** In texture entailment task, we first separately encode the premise and hypothesis sentences and get the sentence representations and then combine the two representations as one to represent the relationship between the two sentences. The feature combining function is defined as

$$\boldsymbol{r} = [\boldsymbol{r}_p; \boldsymbol{r}_h; |\boldsymbol{r}_p - \boldsymbol{r}_h|; \boldsymbol{r}_p * \boldsymbol{r}_h], \tag{10}$$

where $\boldsymbol{r}$ is the final representation of the relation between the premise sentence and hypothesis sentence, $\boldsymbol{r}_p$ is the premise sentence representation and $\boldsymbol{r}_h$ is the hypothesis sentence representation. The schematic of NLI architecture is shown in Figure 3.

Having obtained the representations, we feed the representation vector through a feed forward neural network followed with a softmax normalization function to get the probability vector. Then we optimize a following cross-entropy objective function:

$$\mathcal{J}(\Theta) = \frac{1}{N}\sum_{i=1}^{N} \log p(Y_i; \Theta), \tag{11}$$

where $\Theta$ is trainable parameters in the model, $Y_i$ is the target label of $i$th example in the batch.

## 5 Experiments

We have empirically conducted experiments on 5 text classification tasks and a natural language inference task (SNLI). Detailed dataset statistics are shown in Table 1.

### 5.1 Datasets

**SST-1.** Stanford sentiment Treebank is a movie review dataset which has been parsed and further splited to train/dev/test set [2]. For each example in the dataset, there exists only one sentence and a

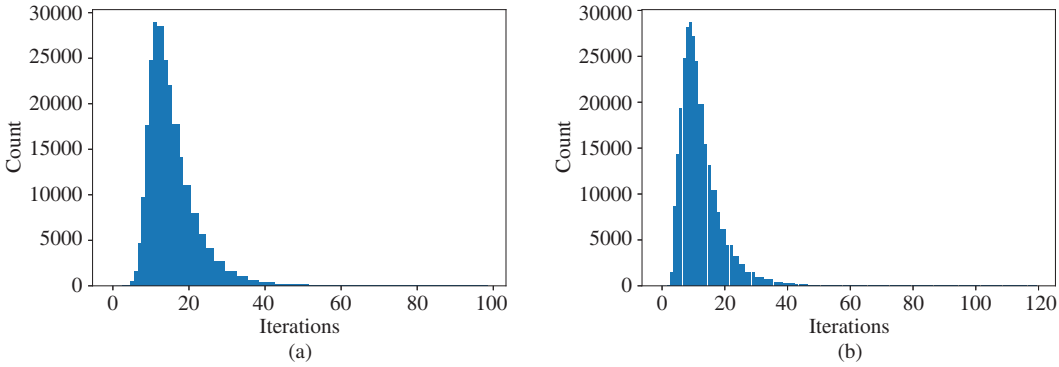


**Figure 4** (Color online) Power method convergence statistics for the SNLI set. It shows that for the majority of samples the power method converges before 200 steps. Note that the vertical axis denotes the sample count and the horizontal axis denotes the iteration steps needed to reach convergence. (a) Premise; (b) hypothesis.

label associated with it. And the labels can be one of {negative, somewhat negative, neutral, somewhat positive, positive}.

**SST-2.** This dataset is a binary-class version of SST-1, with neutral reviews removed and the remaining reviews categorized to either negative or positive.

**IMDB.** IMDB is a movie review dataset extracted from IMDB website. It is a multi-sentence dataset that for each example there are several review sentences. A rating score range from 1 to 10 is also associated with each example.

**Yelp reviews.** Yelp-2013 and Yelp-2014 are reviews from Yelp, each example consists of several review sentences and a rating score range from 1 to 5 (higher is better). Note that we use the same document-level datasets as provided in [1]

**SNLI.** The SNLI dataset is a collection of 570k human-written sentence pairs which are called premise and hypothesis sentence, for each sentence pair there exists a relationship labeled as entailment, contradiction and neutral, and the objective of this task is to predict the relationship between premise and hypothesis sentence. Note that in this paper we encode sentences without interaction.

### 5.2 Implementation details

We adopt adam optimizer [22] to optimize all trainable parameters. To prevent over fitting, we also adopt regularization tricks such as weight decay, dropout are applied on the word embeddings and in the hidden layers of the final feed forward neural net.

The number of hidden units of the connectivity function $f$ in (6) is set to 30 for SNLI dataset, for classification tasks it is set to 50. The word embedding dimensions are unanimously set to 300, the hidden units of the BiLSTMs are set to $(300 + 300)$. We set the stop factor $\epsilon = 1\mathrm{E} - 10$ and set convergence phase max power iteration to 200 steps, as shown in Figure 4, maximum 200 steps of iteration are more than enough for most of the samples. Note that in convergence phase we do not store any intermediate states, and no gradient is required, after convergence we run extra 20 steps to approximate the gradient. Detailed hyper-parameter setting is given in Table 2.

### 5.3 Experimental results

We evaluate our proposed aggregation method on five text classification datasets and a natural language inference dataset (SNLI). IMDB, Yelp-2013 and Yelp-2014 are document-level datasets, SST-1, SST-2 are sentence-level datasets, SNLI is the natural language inference dataset. Because average pooling, self-attention, max-pooling are most related to our study, we mainly compare our study to those baselines, and compare our result on SNLI to encoder based models.

The detailed comparison on classification datasets is shown in Table 3, which shows that our proposed aggregation outperforms all methods in comparison. And specifically a considerable improvement has been achieved on IMDB dataset, 3.7% absolute accuracy improvement compared to DR-AGG.

Detailed comparison result on SNLI is shown in Table 4, which says that our proposed eigen-centrality attention is effective and out-performed max-pooling, mean-pooling and self-attention.

**Table 2** Detailed hyper-parameter settings

| | Yelp-2013 | Yelp-2014 | IMDB | SST-1 | SST-2 | SNLI |
|---|---|---|---|---|---|---|
| Embedding size | 300 | 300 | 300 | 300 | 300 | 300 |
| LSTM hidden unit | 300 | 300 | 300 | 300 | 300 | 300 |
| Connectivity hidden units | 50 | 50 | 50 | 50 | 50 | 30 |
| Regularization rate | 1E−6 | 1E−6 | 1E−6 | 1E−6 | 1E−6 | 1E−20 |
| Initial learning rate | 0.0001 | 0.0001 | 0.0001 | 0.0003 | 0.0003 | 0.0001 |
| Learning rate decay | 0.9 | 0.9 | 0.9 | 0.95 | 0.95 | 0.95 |
| Learning rate decay steps | 2000 | 5000 | 1000 | 500 | 500 | 20000 |
| Initial batch size | 64 | 64 | 32 | 128 | 128 | 128 |
| Batch size low bound | 32 | 16 | 32 | 32 | 16 | 128 |
| Dropout rate | 0.6 | 0.6 | 0.4 | 0.6 | 0.6 | 0.2 |

**Table 3** Experimental results comparison on five classification datasets

| | Yelp-2013 | Yelp-2014 | IMDB | SST-1 | SST-2 |
|---|---|---|---|---|---|
| RNTN+Recurrent [2] | 57.4 | 58.2 | 40.0 | – | – |
| CNN-non-static [5] | – | – | – | 48.0 | 87.2 |
| Paragraph-Vec [23] | – | – | – | 48.7 | 87.8 |
| MT-LSTM (F2S) [4] | – | – | – | 49.1 | 87.2 |
| UPNN (np UP) [1] | 57.7 | 58.5 | 40.5 | – | – |
| UPNN (full) [1] | 59.6 | 60.8 | 43.5 | – | – |
| Cached LSTM [24] | 59.4 | 59.2 | 42.1 | – | – |
| Standard DR-AGG [7] | 62.1 | 63.0 | 45.1 | 50.5 | 87.6 |
| Reverse DR-AGG [7] | 61.6 | 62.5 | 44.5 | 49.3 | 87.2 |
| Max pooling | 61.1 | 61.2 | 41.1 | 48.0 | 87.0 |
| Average pooling | 60.7 | 60.6 | 39.1 | 46.2 | 85.2 |
| Self-attention | 61.0 | 61.5 | 43.3 | 48.2 | 86.4 |
| This work | **63.7** | **64.2** | **48.2** | **51.6** | **88.5** |

**Table 4** Experimental results comparison on SNLI

| Model | Test accuracy |
|---|---|
| Max pooling [25] | 84.5 |
| Intra-attention [26] | 84.2 |
| Self-attention [6] | 84.4 |
| Average pooling | 83.4 |
| This work | **85.3** |

### 5.4 Visualization of latent graph

One important feature in our study is that, we have an intermediate graph, and from this graph we can apply the power method to get the eigen-centrality measure of each node in this graph. A sample from SNLI dataset is shown in Figure 5, where Figure 5(a) is the premise sentence and Figure 5(b) is the hypothesis sentence. The color density of words indicates the importance of each word in this sentence, and the edge is the connection between words, a darker color indicates a more intense connection. Note that we first have the connection and then derive the importance measure, not the other way around. In the example, the word "rover" has the most attention, and the attention is attributed to the nodes which are connected to it, if the node that connects to it is important and has an intense connection, then the node contributes more to the word. This distinguishes eigen-centrality attention from almost all existing aggregation methods, which have been identified as a weak contextual dependency as described in Section 1.

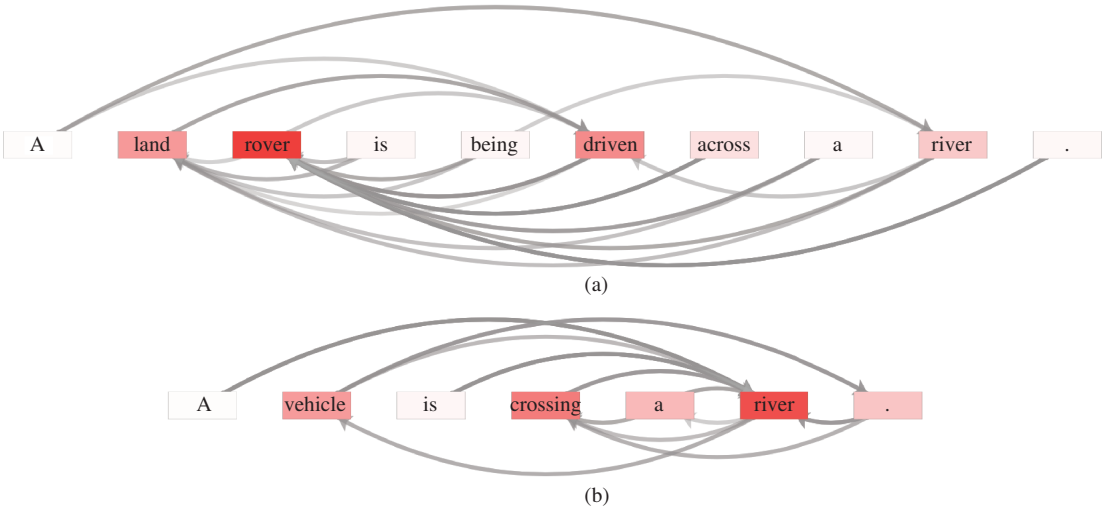


**Figure 5** (Color online) Latent graph visualization of a sample in SNLI dataset. (a) is the premise sentence and (b) is the hypothesis sentence. The color density of words indicates the importance of each word in this sentence, and the edge is the connection between words, a darker color indicates a more intense connection.

## 6 Discussion

### 6.1 Connection to self-attention

The self-attention is actually a subspace inside the space of eigen-centrality attention. Eq. (6) could be optimized to a subspace so that it only cares about $\boldsymbol{h}_i$, then

$$A_{ij} = g(\boldsymbol{h}_i). \tag{12}$$

If the function $f$ is collapsed into $g$ and $g$ is a linear function as in self-attention, with a column softmax normalization, the eigenvector is exactly the same as the assigned weights in self-attention.

In self-attention, the score of a component depends on the component itself (if not consider the final normalization), while in eigen-centrality attention, the score of a component has a recursive dependency on all components connected to it.

### 6.2 Connection to dynamic routing

Dynamic routing [7] and eigen-centrality attentions share the same attribute, that is, which source component is to pass more information to the target is determined globally by the source components. In dynamic routing for aggregation, the route is iteratively updated, the more iterations have passed, the more global information are considered. While in eigen-centrality attention, the weight of each component is considered with a ranking procedure via computing the eigen-centrality.

## 7 Conclusion

In this study, we have proposed a new approach of aggregation, which we call eigen-centrality self-attention. Instead of computing the attention score for each word with a sole dependency on the word itself as done in the vanilla self-attention, eigen-centrality self-attention computes a score for each word and takes the whole sequence into consideration. It achieved by building a fully-connected word graph, in which the connected relationships are described by an adjacency matrix. We can compute the eigen-centrality measure of each word in the graph which indicates how important it is in the graph. Thus it will output attention score for each word with global information considered. Because we apply the power method to obtain the eigenvector, it is non-trivial to compute the gradient for the power method operation, we can compute the gradient with auto-differentiation tools, but this brings other problems such as large memory consumption and considerable computation overhead. Thus we introduced two ways of computing gradient with respect to the adjacency matrix. The first is to run extra iterations just for the step without storing intermediate states in convergence phase, and this reduces considerable

amount of memory consumption and some computation consumption. The second way is to compute gradient after convergence phase analytically without storing any intermediate states (no extra gradient iterations needed), and this reduces memory and computation consumption even more. Also note that because we are taking an iterative approach for the eigenvector, it is expectable that the approach is time demanding, especially in the case where the sentence is longer. When experimenting on Yelp2014, we observed that 5 times more time are required than that of attentive pooling.

When applying eigen-centrality self-attention to encode a sentence, we have the byproduct — the word graph is also very interesting. In the future we expect to drive our aggregation approach with more supervised data such as large parallel translation datasets and devise some unsupervised task if possible, and expect to see some more interesting graph patterns in natural language.

**Acknowledgements** This work was supported by National Natural Science Foundation of China (Grant Nos. 61751201, 61672162), Shanghai Municipal Science and Technology Major Project (Grant No. 2018SHZDZX01), and ZJLab.

**References**

1 Tang D Y, Qin B, Liu T. Learning semantic representations of users and products for document level sentiment classification. In: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, 2015. 1014–1023
2 Socher R, Perelygin A, Wu J, et al. Recursive deep models for semantic compositionality over a sentiment treebank. In: Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, 2013. 1631–1642
3 Bowman S R, Angeli G, Potts C, et al. A large annotated corpus for learning natural language inference. In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, 2015. 632–642
4 Liu P F, Qiu X P, Chen X C, et al. Multi-timescale long short-term memory neural network for modelling sentences and documents. In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, 2015. 2326–2335
5 Kim Y. Convolutional neural networks for sentence classification. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2014. 1746–1751
6 Lin Z H, Feng M W, Santos C N, et al. A structured self-attentive sentence embedding. In: Proceedings of the 5th International Conference on Learning Representations, 2017
7 Gong J J, Qiu X P, Wang S J, et al. Information aggregation via dynamic routing for sequence encoding. In: Proceedings of the 27th International Conference on Computational Linguistics, 2018
8 Yang Z C, Yang D Y, Dyer C, et al. Hierarchical attention networks for document classification. In: Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, 2016. 1480–1489
9 Mikolov T, Sutskever I, Chen K, et al. Distributed representations of words and phrases and their compositionality. In: Proceedings of Advances in Neural Information Processing Systems, 2013. 3111–3119
10 Pennington J, Socher R, Manning C. Glove: global vectors for word representation. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2014. 1532–1543
11 Peters M E, Neumann M, Iyyer M, et al. Deep contextualized word representations. In: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, 2018. 2227–2237
12 Sabour S, Frosst N, Hinton G E. Dynamic routing between capsules. In: Proceedings of Advances in Neural Information Processing Systems, 2017. 3856–3866
13 Zhang X S, Li P S, Jia W J, et al. Multi-labeled relation extraction with attentive capsule network. In: Proceedings of the 33rd AAAI Conference on Artificial Intelligence, 2019
14 Fletcher J M K, Wennekers T. From structure to activity: using centrality measures to predict neuronal activity. Int J Neur Syst, 2018, 28: 1750013
15 Bonacich P. Factoring and weighting approaches to status scores and clique identification. J Math Soc, 1972, 2: 113–120
16 Bonacich P. Some unique properties of eigenvector centrality. Soc Netw, 2007, 29: 555–564
17 Lohmann G, Margulies D S, Horstmann A, et al. Eigenvector centrality mapping for analyzing connectivity patterns in fMRI data of the human brain. Plos One, 2010, 5: e10232
18 Roffo G, Melzi S. Features selection via eigenvector centrality. In: Proceedings of New Frontiers in Mining Complex Patterns, 2016
19 Frobenius V G. Über Matrizen Aus Nicht Negativen Elementen. 1912. 456–477
20 Perron O. Zur theorie der matrices. Math Ann, 1907, 64: 248–263
21 Lin F, Cohen W W. Power iteration clustering. In: Proceedings of the 27th International Conference on Machine Learning ICML-10, 2010. 655–662
22 Kingma D P, Ba J. Adam: a method for stochastic optimization. In: Proceedings of the 3rd International Conference on Learning Representations, 2015
23 Le Q, Mikolov T. Distributed representations of sentences and documents. In: Proceedings of International Conference on Machine Learning, 2014. 1188–1196
24 Xu J C, Chen D L, Qiu X P, et al. Cached long short-term memory neural networks for document-level sentiment classification. In: Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, 2016. 1660–1669
25 Conneau A, Kiela D, Schwenk H, et al. Supervised learning of universal sentence representations from natural language inference data. In: Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, 2017. 670–680
26 Liu Y, Sun C J, Lin L, et al. Learning natural language inference using bidirectional LSTM model and inner-attention. 2016. ArXiv: 1605.09090

# Appendix A

## Appendix A.1 Gradient of power method

Given an adjacency matrix $\boldsymbol{A}$ with all its components to be strictly positive, we can obtain a good approximation of dominant eigenvector $\boldsymbol{\alpha}$ and its corresponding eigenvalue $\lambda$ via power method. Now we need to compute the partial derivative of $\boldsymbol{\alpha}$ with respect to the eigenvector $\boldsymbol{A}$, the power method can be formulated as follows:

$$\boldsymbol{\alpha}^t = \frac{\boldsymbol{A}^{t-1}\boldsymbol{\alpha}^{t-1}}{\|\boldsymbol{A}^{t-1}\boldsymbol{\alpha}^{t-1}\|_2}, \tag{A1}$$

where $\boldsymbol{\alpha}^t$ is the $t$th step of eigenvector, and $\boldsymbol{A}^{t-1}$ is the $(t-1)$th adjacency matrix, the value between $\boldsymbol{A}^t$ and $\boldsymbol{A}^{t-1}$ is the same, because the power method is an iterative approach, we need to consider the gradient for each step and summarize them in the end. Also note that when converged and with $t$ large enough, $\boldsymbol{\alpha}^t \approx \boldsymbol{\alpha}^{t-1}$ and $\|\boldsymbol{A}^{t-1}\boldsymbol{\alpha}^{t-1}\|_2 \approx \|\boldsymbol{A}\boldsymbol{\alpha}\|_2 \approx \lambda$.

Assuming $t$ is large enough and iteration long converged, first inspect the partial derivative of $\boldsymbol{\alpha}_p^t$ with respect to $A_{qr}^{t-1}$ to obtain the jacobian of $\boldsymbol{\alpha}^t$ with respect to $\boldsymbol{A}^{t-1}$. For simplicity, we denote $\boldsymbol{A}^{t-1}$ as $\boldsymbol{M}$, and because we need only the value of $\boldsymbol{\alpha}^t$ and $\boldsymbol{\alpha}^{t-1}$, we denote them as $\boldsymbol{\alpha}$:

$$\alpha_p = \frac{\sum_i M_{pi}\alpha_i}{\|\boldsymbol{M}\boldsymbol{\alpha}\|_2}, \tag{A2}$$

$$\frac{\partial \alpha_p}{\partial M_{qr}} = \frac{\frac{\partial \sum_i M_{pi}\alpha_i}{\partial M_{qr}}\|\boldsymbol{M}\boldsymbol{\alpha}\|_2 - \frac{\partial \|\boldsymbol{M}\boldsymbol{\alpha}\|_2}{\partial M_{qr}}\sum_i M_{pi}\alpha_i}{(\|\boldsymbol{M}\boldsymbol{\alpha}\|_2)^2}. \tag{A3}$$

Replace $\|\boldsymbol{M}\boldsymbol{\alpha}\|_2$ with $\lambda$, and $\sum_i M_{pi}\alpha_i$ with $\lambda\alpha_p$:

$$\frac{\partial \alpha_p}{\partial M_{qr}} = \frac{\lambda\frac{\partial \sum_i M_{pi}\alpha_i}{\partial M_{qr}} - \lambda\alpha_p\frac{\partial \|\boldsymbol{M}\boldsymbol{\alpha}\|_2}{\partial M_{qr}}}{\lambda^2}, \tag{A4}$$

$$\frac{\partial \sum_i M_{pi}\alpha_i}{\partial M_{qr}} = \mathbb{1}_{p=q}\alpha_r, \tag{A5}$$

$$\frac{\partial \|\boldsymbol{M}\boldsymbol{\alpha}\|_2}{\partial M_{qr}} = \frac{1}{2\|\boldsymbol{M}\boldsymbol{\alpha}\|_2}\frac{\partial \sum_i (\sum_j M_{ij}\alpha_j)^2}{\partial M_{qr}} \tag{A6}$$

$$= \frac{1}{2\lambda}\cdot 2\left(\sum_j M_{qj}\alpha_j\right)\alpha_r \tag{A7}$$

$$= \alpha_q\alpha_r. \tag{A8}$$

Plug (A5) and (A8) into (A4):

$$\frac{\partial \alpha_p}{\partial M_{qr}} = \frac{\mathbb{1}_{p=q}\alpha_r - \alpha_p\alpha_q\alpha_r}{\lambda}. \tag{A9}$$

We can gather the partial derivative $\frac{\partial \alpha_p}{\partial M_{qr}}$ to a jacobian matrix $\boldsymbol{J}_{\boldsymbol{A}} \in \mathbb{R}^{n\times n\times n}$.

We then inspect the partial derivative of $\alpha_p^t$ with respect to $\alpha_q^{t-1}$. Also we can denote $\boldsymbol{A}^{t-1}$ as $\boldsymbol{M}$:

$$\alpha_p^t = \frac{\sum_i M_{pi}v_i^{t-1}}{\|\boldsymbol{M}\boldsymbol{\alpha}\|_2} \tag{A10}$$

$$\frac{\partial \alpha_p^t}{\partial \alpha_q^{t-1}} = \frac{\frac{\partial \sum_i M_{pi}v_i^{t-1}}{\partial v_q^{t-1}}\|\boldsymbol{M}\boldsymbol{\alpha}^{t-1}\|_2 - \frac{\partial \|\boldsymbol{M}\boldsymbol{\alpha}^{t-1}\|_2}{\partial v_q^{t-1}}\sum_i M_{pi}v_i^{t-1}}{(\|\boldsymbol{M}\boldsymbol{\alpha}^{t-1}\|_2)^2}. \tag{A11}$$

Replace $\|\boldsymbol{M}\boldsymbol{\alpha}^{t-1}\|_2$ with $\lambda$, and $\sum_i M_{pi}v_i^{t-1}$ with $\lambda\alpha_p^{t-1}$:

$$\frac{\partial \alpha_p^t}{\partial \alpha_q^{t-1}} = \frac{\lambda\frac{\partial \sum_i M_{pi}v_i^{t-1}}{\partial \alpha_q^{t-1}} - \lambda\alpha_p^{t-1}\frac{\partial \|\boldsymbol{M}\boldsymbol{\alpha}\|_2}{\partial \alpha_q^{t-1}}}{\lambda^2}, \tag{A12}$$

$$\frac{\partial \sum_i M_{pi}v_i^{t-1}}{\partial \alpha_q^{t-1}} = M_{pq}, \tag{A13}$$

$$\frac{\partial \|\boldsymbol{M}\boldsymbol{\alpha}\|_2}{\partial \alpha_q^{t-1}} = \frac{1}{2\|\boldsymbol{M}\boldsymbol{\alpha}\|_2}\frac{\partial \sum_i (\sum_j M_{ij}\alpha_j)^2}{\partial \alpha_q^{t-1}} \tag{A14}$$

$$= \frac{1}{2\lambda} \cdot 2 \sum_i M_{iq} \sum_j M_{ij} v_j^{t-1} \tag{A15}$$

$$= \frac{1}{2\lambda} \cdot 2\lambda \sum_i M_{iq} v_i^{t-1} \tag{A16}$$

$$= (\boldsymbol{\alpha}^{t-1})^{\mathrm{T}} \boldsymbol{M}_{\cdot q}. \tag{A17}$$

Plug (A13) and (A17) into (A12):

$$\frac{\partial \alpha_p^t}{\partial \alpha_q^{t-1}} = \frac{M_{pq} - \alpha_p^{t-1} (\boldsymbol{\alpha}^{t-1})^{\mathrm{T}} \boldsymbol{M}_{\cdot q}}{\lambda}. \tag{A18}$$

Denote $\frac{\partial \alpha_p^t}{\partial \alpha_q^t}$ as $J_{pq}$, and because $\boldsymbol{\alpha}^t \approx \boldsymbol{\alpha}^{t-1}$, for simplicity we denote them as $\boldsymbol{\alpha}$:

$$J_{pq} = \frac{M_{pq} - \alpha_p \boldsymbol{\alpha}^{\mathrm{T}} \boldsymbol{M}_{\cdot q}}{\lambda}, \tag{A19}$$

rearrange $J_{pq}$ to a jacobian matrix $\boldsymbol{J}_{\boldsymbol{\alpha}} \in \mathbb{R}^{n \times n}$:

$$\boldsymbol{J}_{\boldsymbol{\alpha}} = \frac{\boldsymbol{M} - \boldsymbol{\alpha}\boldsymbol{\alpha}^{\mathrm{T}} \boldsymbol{M}}{\lambda}. \tag{A20}$$

Finally we can apply chain rule to get the gradient of loss $\mathcal{L}$ with respect to adjacency matrix $\boldsymbol{A}$. Assuming that we have a gradient from loss $\mathcal{L}$ with respect to the converged eigenvector:

$$\boldsymbol{\gamma} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{\alpha}} \in \mathbb{R}^n, \tag{A21}$$

$$\frac{\partial L}{\partial \boldsymbol{A}} = \lim_{N \to \infty} \sum_{k=0}^{N} \boldsymbol{\gamma}^{\mathrm{T}} (\boldsymbol{J}_{\boldsymbol{\alpha}})^k \boldsymbol{J}_{\boldsymbol{A}}. \tag{A22}$$

Perron-Frobenius theorem [19, 20] says that

$$\lim_{N \to \infty} \frac{\boldsymbol{M}^N}{\lambda^N} = \boldsymbol{\alpha}\boldsymbol{w}^{\mathrm{T}}, \tag{A23}$$

where $\boldsymbol{\alpha}$ is the right dominant eigenvector and $\boldsymbol{w}$ is the left dominant eigenvector, $\boldsymbol{\alpha}$ is normalized so that $\|\boldsymbol{\alpha}\|_2 = 1$ and $\boldsymbol{w}$ is normalized so that $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\alpha} = 1$. Then,

$$\lim_{N \to \infty} (\boldsymbol{J}_{\boldsymbol{\alpha}})^N = \lim_{N \to \infty} \frac{\boldsymbol{M}^N - \boldsymbol{\alpha}\boldsymbol{\alpha}^{\mathrm{T}} \boldsymbol{M}^N}{\lambda^N} \tag{A24}$$

$$= \boldsymbol{0}. \tag{A25}$$

Hua et al.[1)] proved the power method converge exponentially by the order of $(\frac{\lambda_2}{\lambda_1})^N$ where $\lambda_1$ is the dominant eigenvector and $\lambda_2$ is the second eigenvector, $0 < \frac{\lambda_2}{\lambda_1} < 1$, which means that Eq. (A24) converges exponentially to 0. Thus the series in (A22) have an upper bound. Also Eq. (A22) tells us that we can take an iterative approach to get the gradient and practically we do not meed much iteration to get a good enough approximation.

## Appendix A.2 Analysis of power method initialization

We prove that the gradient is irrelevant with respect to the randomly initialized vector. We can prove that by proving that the initial vector gets a gradient of zero. Assuming we have run power method for $N$ steps, then the initial vector $\boldsymbol{z}$ receives a gradient as follows:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{z}} = \boldsymbol{\gamma}^{\mathrm{T}} \boldsymbol{J}_{\boldsymbol{\alpha}}^N \tag{A26}$$

by using mathematical induction, we can easily prove that

$$\boldsymbol{J}_{\boldsymbol{\alpha}}^N = \frac{\boldsymbol{M}^N - \boldsymbol{\alpha}\boldsymbol{\alpha}^{\mathrm{T}} \boldsymbol{M}^N}{\lambda^N}, \tag{A27}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{z}} = (\boldsymbol{\gamma}^{\mathrm{T}} - \boldsymbol{\gamma}^{\mathrm{T}} \boldsymbol{\alpha}\boldsymbol{\alpha}^{\mathrm{T}}) \frac{\boldsymbol{M}^N}{\lambda^N}, \tag{A28}$$

$$\lim_{N \to \infty} \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}} = (\boldsymbol{\gamma}^{\mathrm{T}} - \boldsymbol{\gamma}^{\mathrm{T}} \boldsymbol{\alpha}\boldsymbol{\alpha}^{\mathrm{T}}) \lim_{N \to \infty} \frac{\boldsymbol{M}^N}{\lambda^N}. \tag{A29}$$

1) Hua Y B, Xiang Y, Chen T P, et al. A new look at the power method for fast subspace tracking. Digit Signal Process, 1999, 9: 297–314.

Perron-Frobenius theorem [19, 20] says that

$$\lim_{N\to\infty}\frac{\boldsymbol{M}^N}{\lambda^N}=\boldsymbol{\alpha}\boldsymbol{w}^{\mathrm{T}}, \tag{A30}$$

where $\boldsymbol{\alpha}$ is the right dominant eigenvector and $\boldsymbol{w}$ is the left dominant eigenvector, $\boldsymbol{\alpha}$ is normalized so that $\|\boldsymbol{\alpha}\|_2=1$ and $\boldsymbol{w}$ is normalized so that $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\alpha}=1$. Then we have

$$\frac{\partial\mathcal{L}}{\partial\boldsymbol{z}}=(\boldsymbol{\gamma}^{\mathrm{T}}-\boldsymbol{\gamma}^{\mathrm{T}}\boldsymbol{\alpha}\boldsymbol{\alpha}^{\mathrm{T}})\boldsymbol{\alpha}\boldsymbol{w}^{\mathrm{T}} \tag{A31}$$

$$=\boldsymbol{\gamma}^{\mathrm{T}}\boldsymbol{\alpha}\boldsymbol{w}^{\mathrm{T}}-\boldsymbol{\gamma}^{\mathrm{T}}\boldsymbol{\alpha}\boldsymbol{\alpha}^{\mathrm{T}}\boldsymbol{\alpha}\boldsymbol{w}^{\mathrm{T}}=\boldsymbol{0}. \tag{A32}$$

This suggests that if we are to compute the gradient via auto-differentiation tools such as tensorflow, when the convergence is slow, we can stop gradient before converge, and run extra steps just to get the gradients. This could save a substantial amount of memory. When the convergence is quick, the extra gradient steps could also help to accurately approximate the gradient.

The better way of approximating the gradient is shown in (A22). Via this iterative approach, no intermediate state are required to be stored. This makes the memory complexity to be $O(1)$, and extra gradient steps are not needed.

## Appendix A.3 Avoid a perpendicular initialization

Cavender[2)] and Hogben[3)] said that even when the adjacency matrix is defective, with its dominant eigenvalue to have algebraic and geometric multiplicities both to be one, the power method would still converge. In this section, we suppose that the adjacency matrix $\boldsymbol{A}$ is a non-defective matrix. And justify the choice of initializing the vector $\boldsymbol{z}$ to have all positive components.

Assuming that strictly positive adjacency matrix $\boldsymbol{A}$ has a full set of basis eigenvectors $\boldsymbol{u}_1,\boldsymbol{u}_2,\ldots,\boldsymbol{u}_n$ and corresponding eigenvalues $\lambda_1,\lambda_2,\ldots,\lambda_n$ with $\lambda_1$ to be the dominant eigenvalue such that $\lambda_1>|\lambda_2|>\cdots>|\lambda_n|$, there exists real numbers $c_1,c_2,\ldots,c_n$ such that

$$\boldsymbol{z}=c_1\boldsymbol{u}_1+c_2\boldsymbol{u}_2+\cdots+c_n\boldsymbol{u}_n. \tag{A33}$$

Then,

$$\boldsymbol{A}^N\boldsymbol{z}=c_1\boldsymbol{A}^N\boldsymbol{u}_1+c_2\boldsymbol{A}^N\boldsymbol{u}_2+\cdots+c_n\boldsymbol{A}^N\boldsymbol{u}_n \tag{A34}$$

$$=c_1\lambda_1^N\boldsymbol{u}_1+c_2\lambda_2^N\boldsymbol{u}_2+\cdots+c_n\lambda_n^N\boldsymbol{u}_n, \tag{A35}$$

$$\frac{\boldsymbol{A}^N\boldsymbol{z}}{\lambda_1^N}=c_1\boldsymbol{u}_1+c_2\left(\frac{\lambda_2}{\lambda_1}\right)^N\boldsymbol{u}_2+\cdots+c_n\left(\frac{\lambda_n}{\lambda_1}\right)^N\boldsymbol{u}_n. \tag{A36}$$

Because $\forall i,\ \frac{\lambda_i}{\lambda_1}<1$ then $\lim_{N\to\infty}(\frac{\lambda_i}{\lambda_1})^N=0$,

$$\lim_{N\to\infty}\frac{\boldsymbol{A}^N\boldsymbol{z}}{\lambda_1^N}=c_1\boldsymbol{u}_1. \tag{A37}$$

Eq. (A37) shows that when $c_1$ is not zero, the power method would converge to the dominant eigenvector, when $c_1$ is zero, the power method is unable to converge to the dominant eigenvector. But luckily in this paper, the dominant eigenvector is known to have all components to be real positive. This suggests that when we initialize a vector $\boldsymbol{z}$ to have all components to be positive, we can guarantee that $c_1$ to be non-zero and possibly converge faster.

2) Cavender T A. The use of the power method to find dominant eigenvalues of matrices. Dissertation for Master's Degree. Denton: University of North Texas, 1993.

3) Hogben L. Elementary Linear Algebra. Belmont: Thomson Learning, 1987.